Networking protocols and administration

ITV8030

lecture 5 part 2:

NAT, firewalls, iptables

lecture plan

- **NAT**, with different goals:
  - IP pools
  - Migration
  - Masquerading
  - Load balancing
- A few words about **firewalls**
- **Iptables**: hugely popular on Linux

- FOR NAT, using mostly slides from Univ of Virginia / Univ of Toronto
- FOR IPTABLES, mostly slides from Kenneth Shelton

Nat main idea

- Let one IP address (say, 220.31.245.12) be used for a large number of different machines on a local network!
- How is that possible?
- All these machines have internal IP addresses like
  - 10.0.0.1
  - 10.0.0.2
  - Etc
- Local network router replaces each **internal address** with a its **own address + new port number**:
  - 10.0.0.1 is represented as, say, 220.31.245.12:30001
  - 10.0.0.2 is represented as, say, 220.31.245.12:30002
  - Etc

Nat main idea continued

- Actually we need to represent the **internal address + port** with **external address+ port**, like this:
  - 10.0.0.1:80 <-> 220.31.245.12:30001
  - 10.0.0.1:2000 <-> 220.31.245.12:30002
  - 10.0.0.2 :80 <-> 220.31.245.12:30003
  - 10.0.0.2:2000 <-> 220.31.245.12:30004
  - Etc
- The router doing NAT keeps a representation table in memory
- There are enough potentially available port numbers (0...65000) to make this feasible for small and mid-size local networks
- The NAT router must mangle all the **IP fields in IP headers** and also **port fields in TCP/UDP packets**

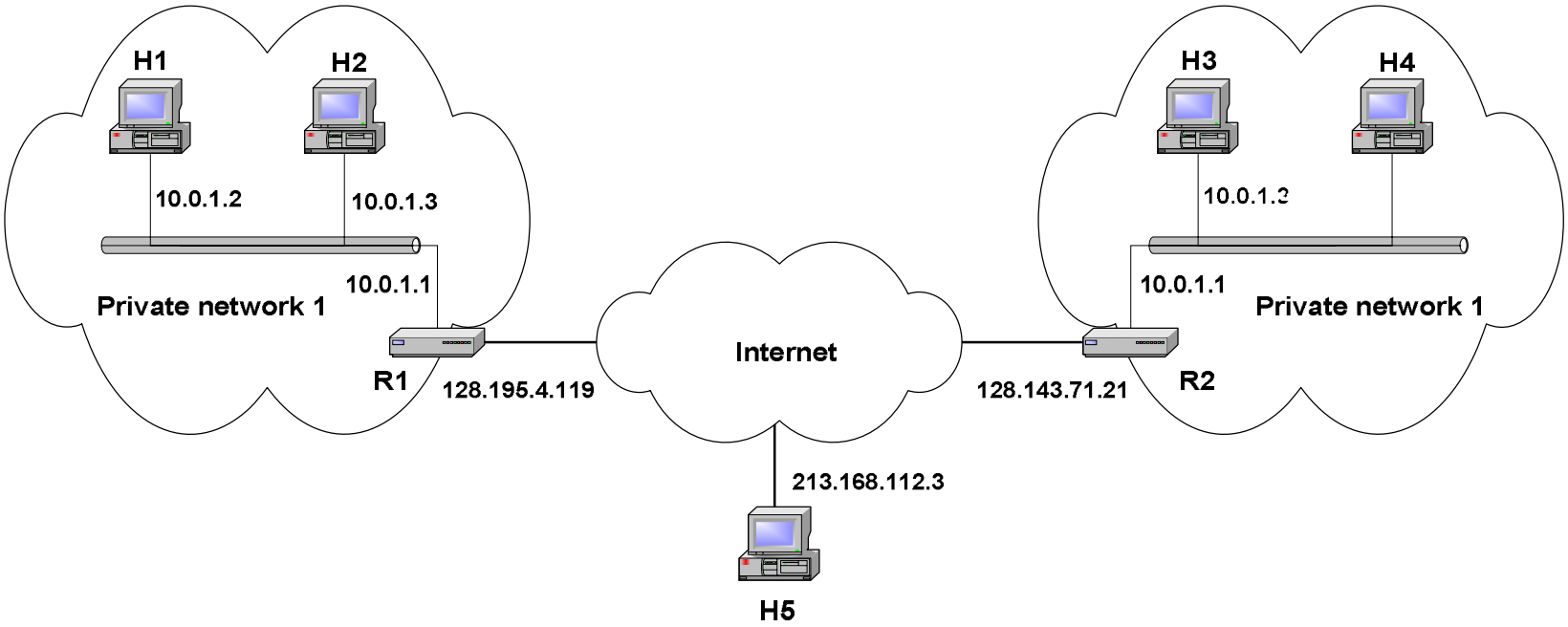
Private Network

- *Private IP* network is an IP network that is not directly connected to the Internet

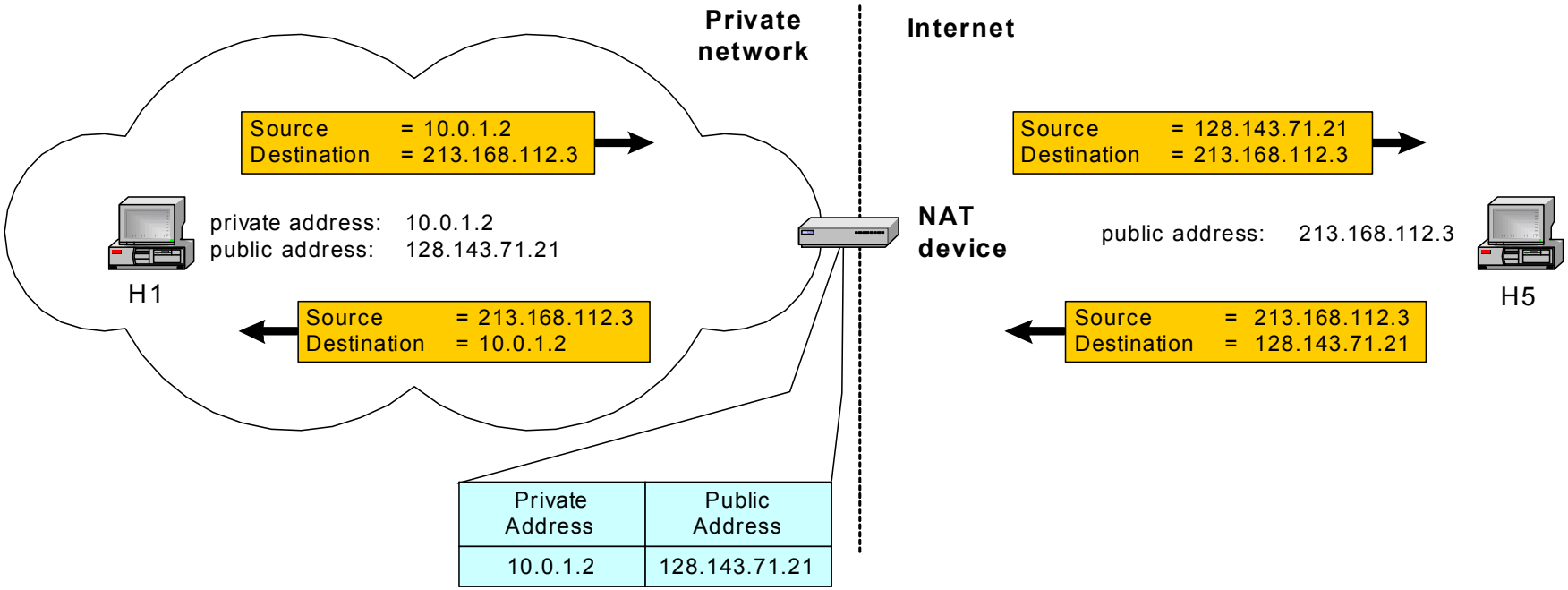
- IP addresses in a private network can be assigned arbitrarily.
  - Not registered and not guaranteed to be globally unique

- Generally, private networks use addresses from the following address ranges (*non-routable addresses*):
  - 10.0.0.0 – 10.255.255.255
  - 172.16.0.0 – 172.31.255.255
  - 192.168.0.0 – 192.168.255.255

Private Addresses



Basic operation of NAT

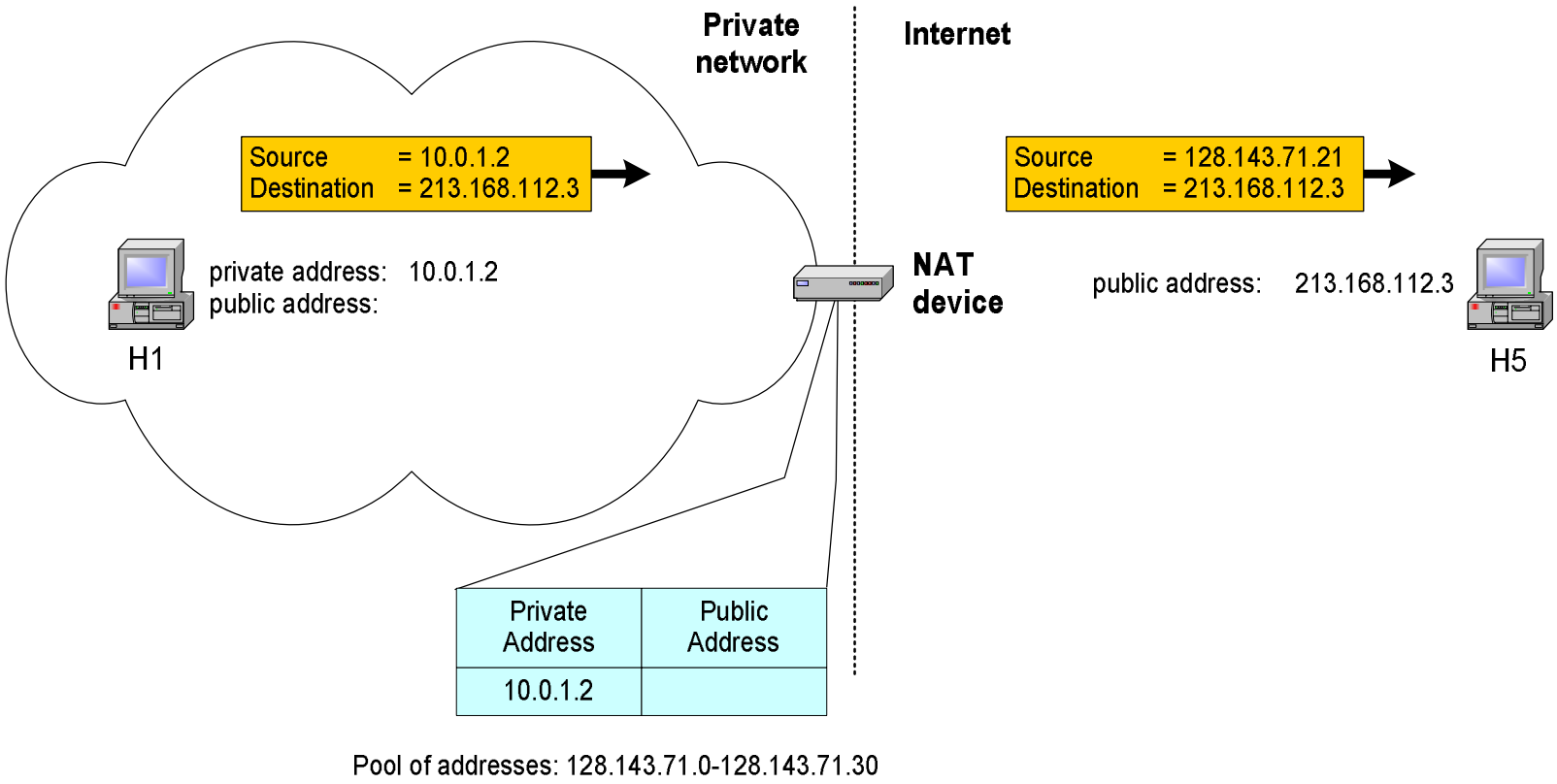


- NAT device has address translation table

Pooling of IP addresses

- **Scenario:** Corporate network has many hosts but only a small number of public IP addresses
- **NAT solution:**
  - Corporate network is managed with a private address space
  - NAT device, located at the boundary between the corporate network and the public Internet, manages a pool of public IP addresses
  - When a host from the corporate network sends an IP datagram to a host in the public Internet, the NAT device picks a public IP address from the address pool, and binds this address to the private address of the host

Pooling of IP addresses



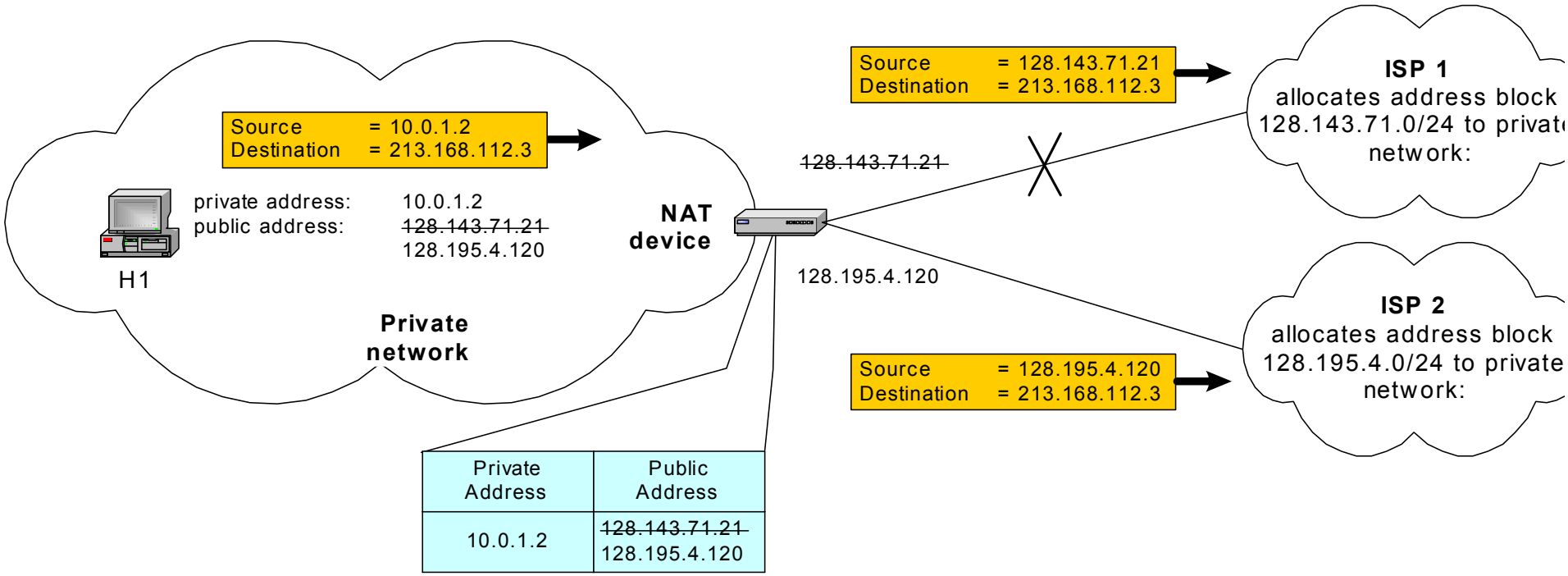
Supporting migration between network service providers

- **Scenario:** In CIDR, the IP addresses in a corporate network are obtained from the service provider. Changing the service provider requires changing all IP addresses in the network.
- **NAT solution:**
  - Assign private addresses to the hosts of the corporate network
  - NAT device has static address translation entries which bind the private address of a host to the public address.
  - Migration to a new network service provider merely requires an update of the NAT device. The migration is not noticeable to the hosts on the network.

  Note:

  - The difference to the use of NAT with IP address pooling is that the mapping of public and private IP addresses is static.

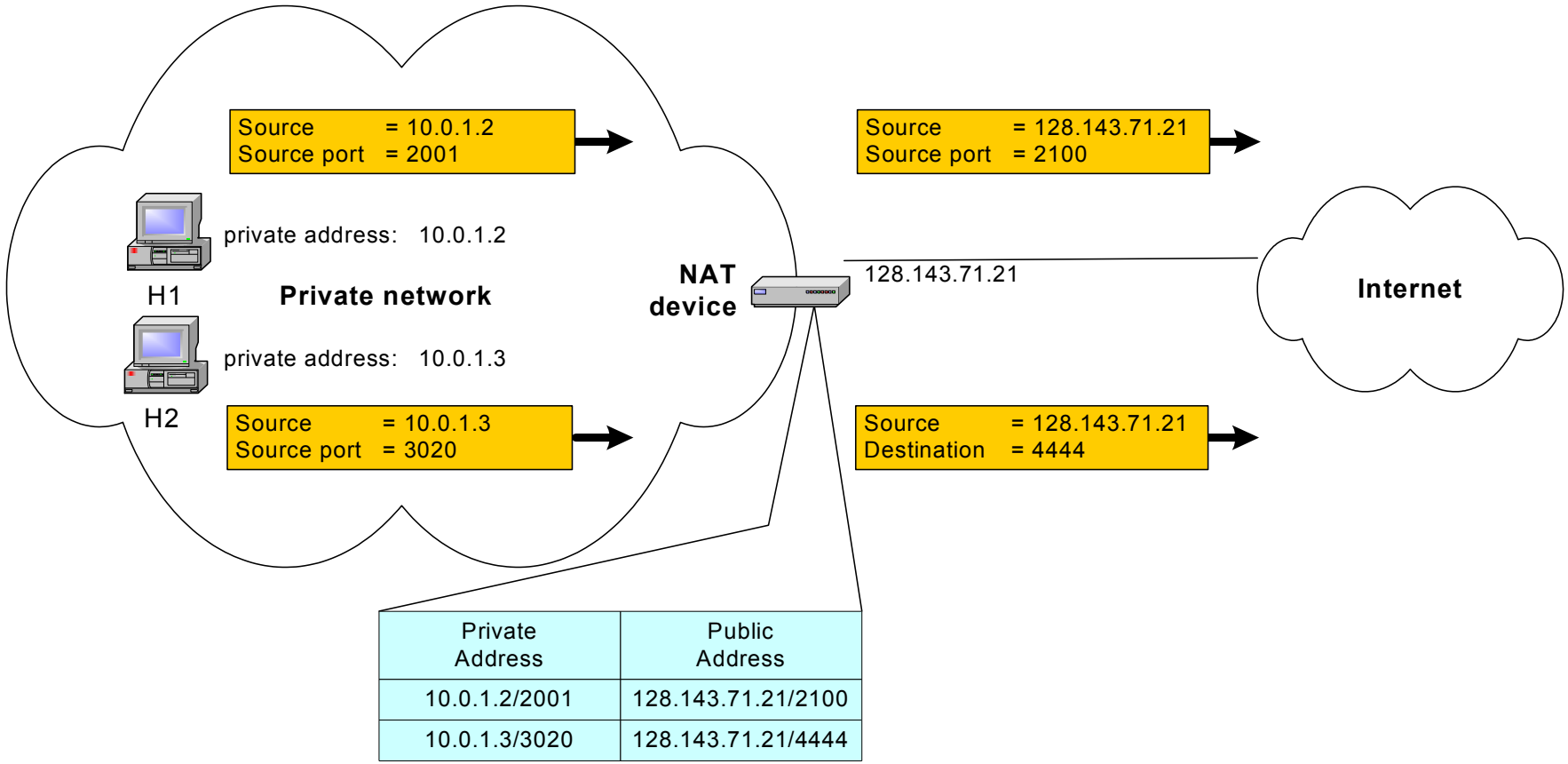
Supporting migration between network service providers



IP masquerading

- **Also called: Network address and port translation (NAPT), port address translation (PAT).**
- **Scenario:** Single public IP address is mapped to multiple hosts in a private network.
- **NAT solution:**
  - Assign private addresses to the hosts of the corporate network
  - NAT device modifies the port numbers for outgoing traffic

IP masquerading

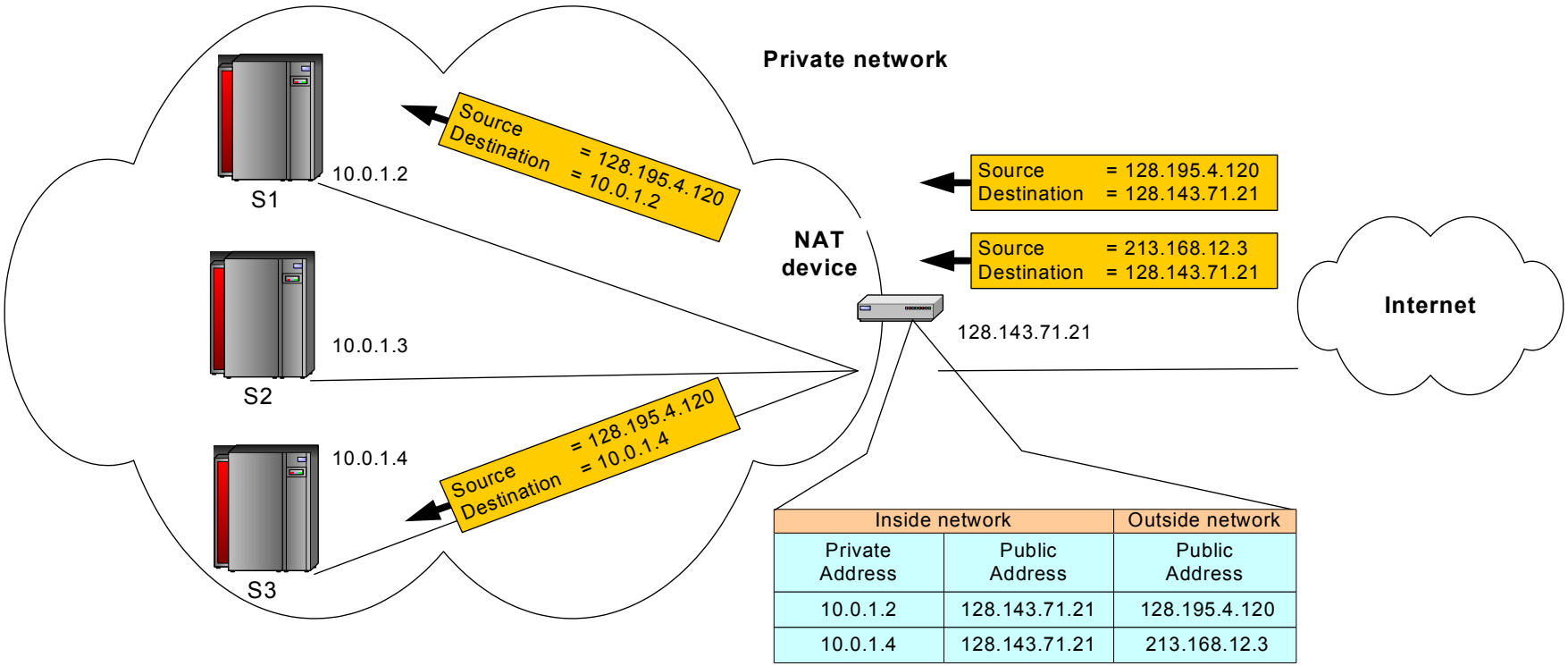


Load balancing of servers

- **Scenario:** Balance the load on a set of identical servers, which are accessible from a single IP address

- **NAT solution:**
  - Here, the servers are assigned private addresses
  - NAT device acts as a proxy for requests to the server from the public network
  - The NAT device changes the destination IP address of arriving packets to one of the private addresses for a server
  - A sensible strategy for balancing the load of the servers is to assign the addresses of the servers in a round-robin fashion.

Load balancing of servers



Inside network		Outside network
Private Address	Public Address	Public Address
10.0.1.2	128.143.71.21	128.195.4.120
10.0.1.4	128.143.71.21	213.168.12.3

Concerns about NAT

- **Performance:**
  - Modifying the IP header by changing the IP address requires that NAT boxes recalculate the IP header checksum
  - Modifying port number requires that NAT boxes recalculate TCP checksum
- **Fragmentation**
  - Care must be taken that a datagram that is fragmented before it reaches the NAT device, is not assigned a different IP address or different port numbers for each of the fragments.

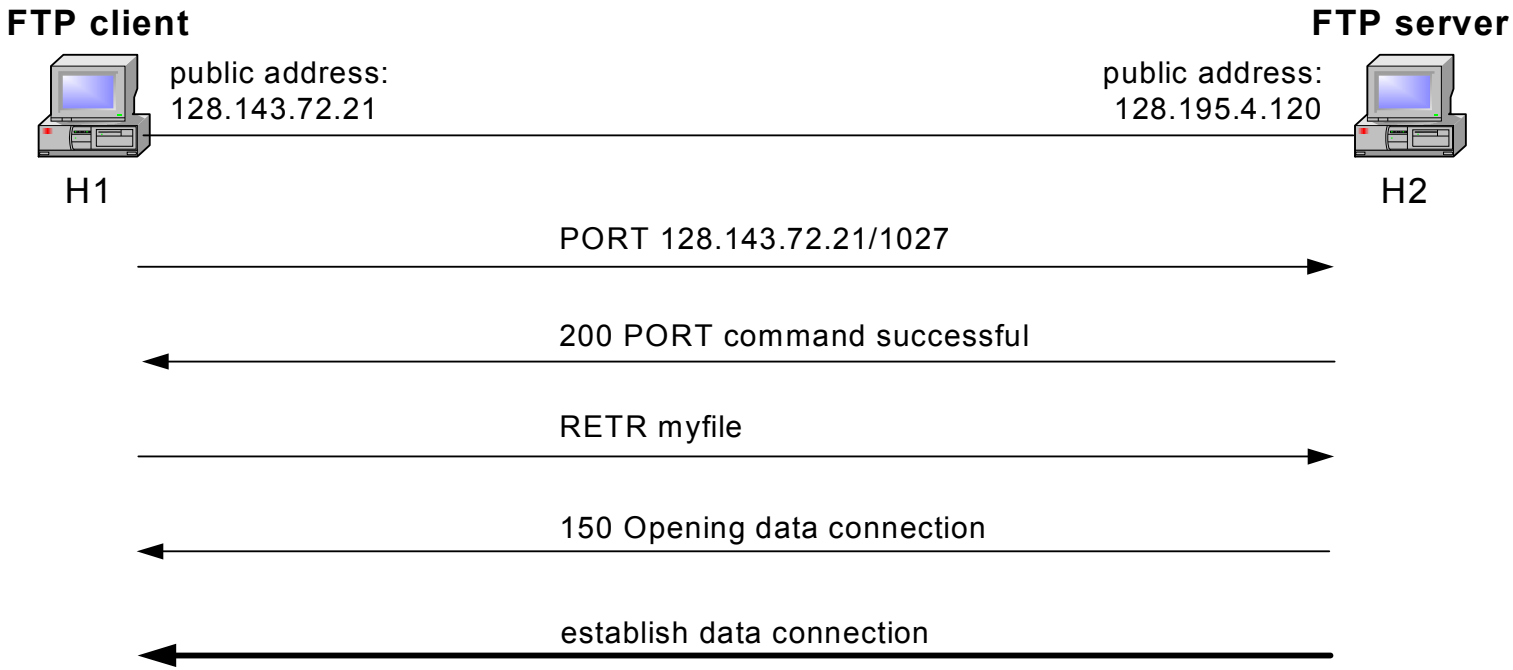
Concerns about NAT

- **End-to-end connectivity:**
  - NAT destroys universal end-to-end reachability of hosts on the Internet.
  - A host in the public Internet often cannot initiate communication to a host in a private network.
  - The problem is worse, when two hosts that are in a private network need to communicate with each other.

Concerns about NAT

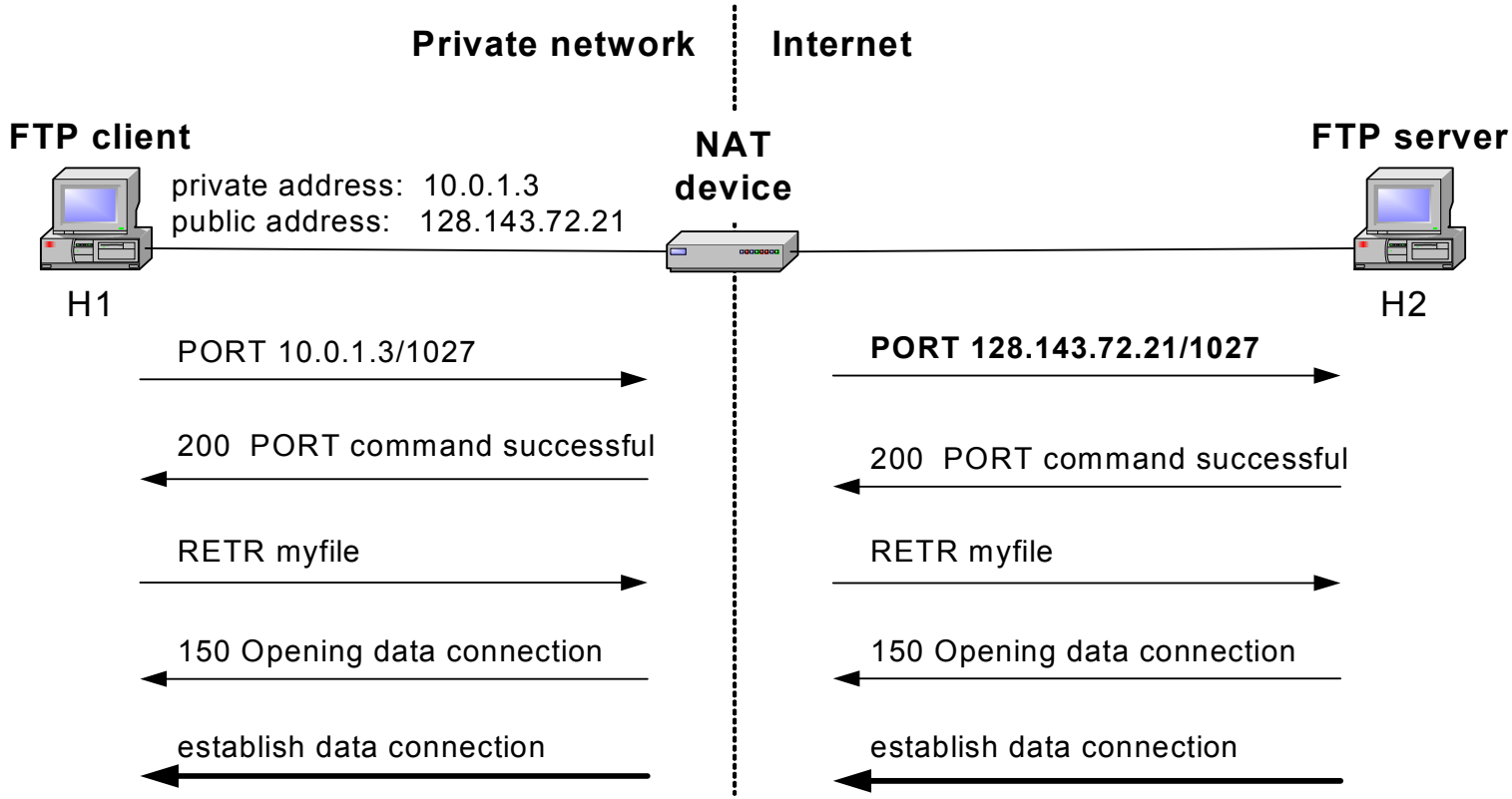
- **IP address in application data:**
  - Applications that carry IP addresses in the payload of the application data generally do not work across a private-public network boundary.
  - Some NAT devices inspect the payload of widely used application layer protocols and, if an IP address is detected in the application-layer header or the application payload, translate the address according to the address translation table.

NAT and FTP



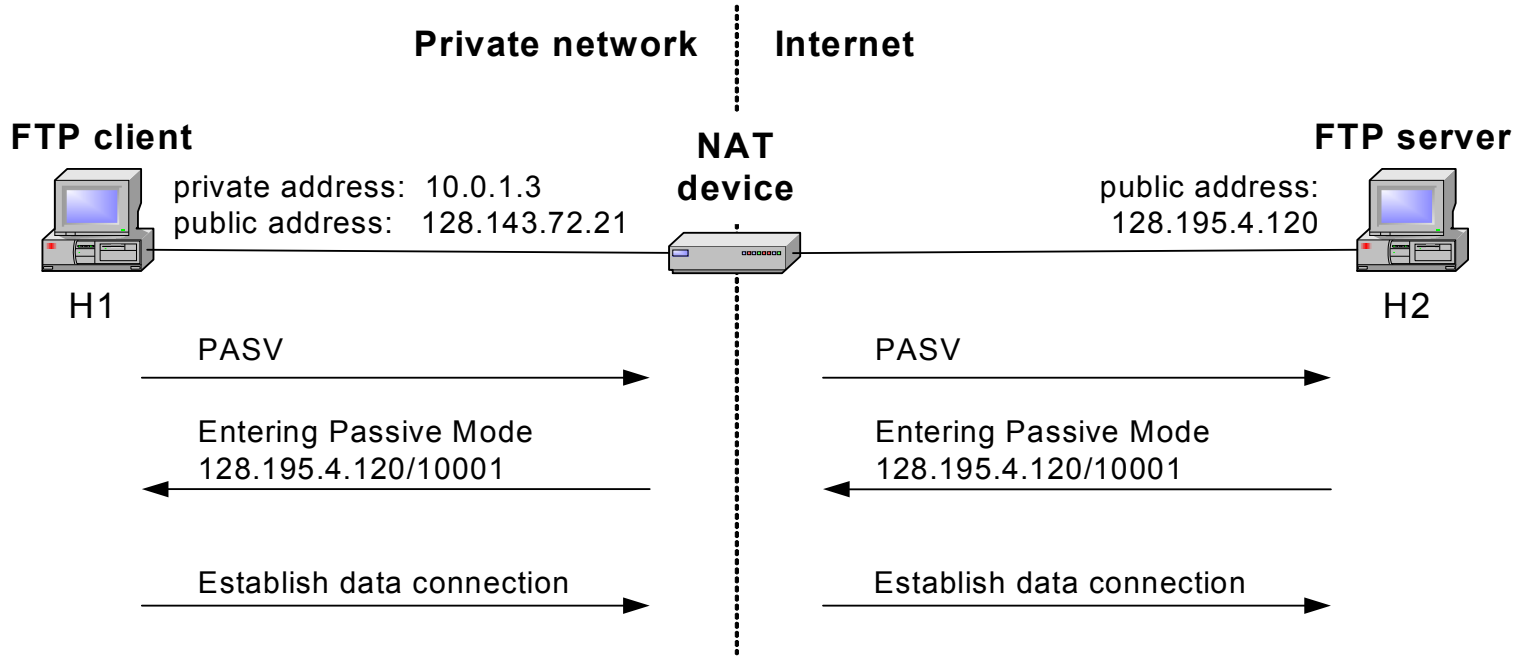
- Normal FTP operation

NAT and FTP



- NAT device with FTP support

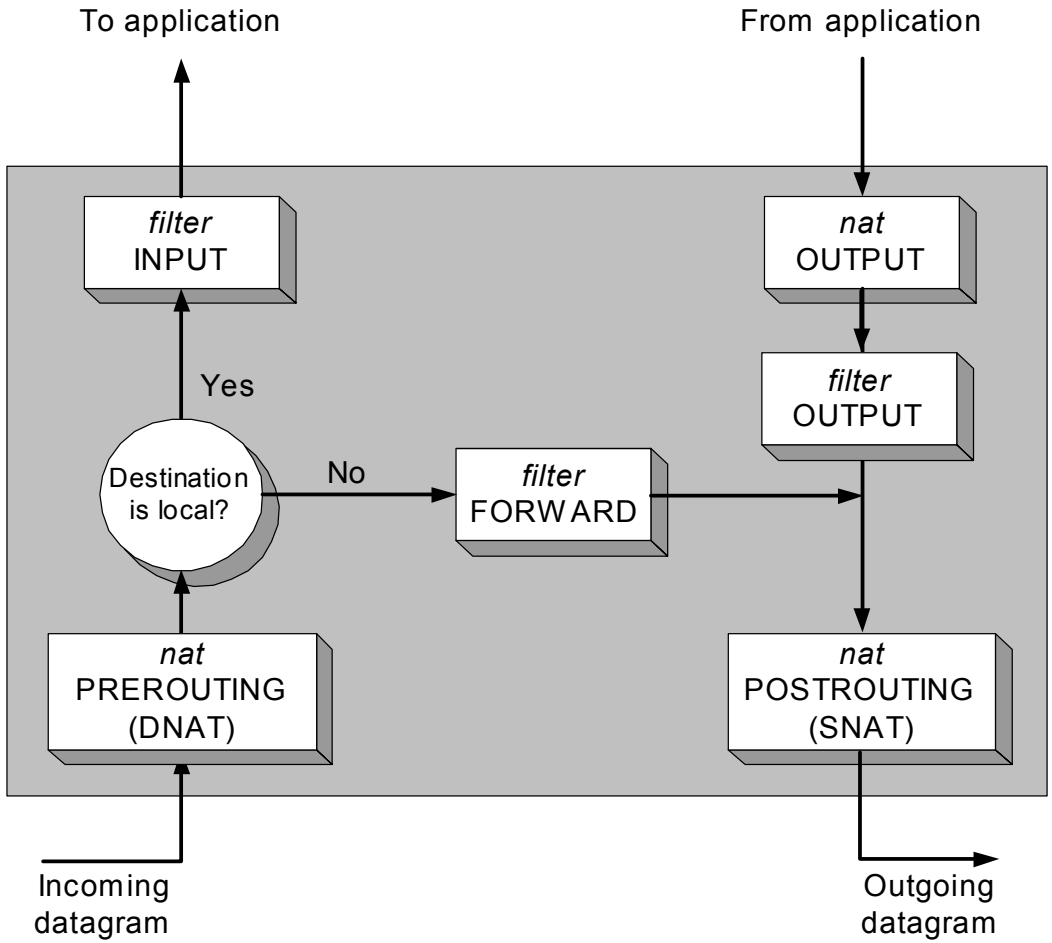
NAT and FTP



- FTP in passive mode and NAT.

Configuring NAT in Linux

- Linux uses the Netfilter/iptable package to add filtering rules to the IP module



FIREWALLS

What is a firewall?

- Basically, the simplest firewall is a filter of packets in front of ordinary applications or other machines
- Why is it useful?
  - It is hard to check that you do not have services running (on some ports) which could be maliciously accessed from outside
  - It is very hard to check that no machine in the internal network has such services
- In other words, block most ports for most machines from one central location

Three generations of firewalls

- Packet filter
  - Just reject some packets (by IP, port or protocol)
- Stateful filter
  - Remember/trace ongoing TCP connections and allow packets inside an existing TCP connection
  - Absolutely critical capability!
- Application-level filter
  - Look into the TCP connection and base drop/pass decision on that.
  - For example, block viruses and trojans etc

Major problem with a packet filter!

- When we do not allow packets to enter our system, we cannot use TCP at all:
  - TCP packets go from us to external server,
  - and vice versa, from external to us
- Say, I open http://www.epl.ee:80
- That is, I will send packets to www.epl.ee to port 80, indicating my IP and (dynamic) port (say, A.B.C.D:30000) where I want to get packets
- Then the www.epl.ee must send packets to A.B.C.D:30000 in return!
- I.E: we cannot really close our ports!!!

Stateful firewall solution!

- Trace all outgoing TCP packets and remember to which IP and port should the external system send TCP packets back
- Possibly check other issues, with the goal to understand whether TCP channel is still open
- Let TCP packets from external server in only for this already-open TCP channel (started from the inside!)

IPTABLES

What Is Netfilter/Iptables?

- Improved successor to ipchains available in linux kernel 2.4/2.6.
- Netfilter is a set of kernel hooks for allowing kernel modules to register callbacks with the network stack.
- Iptables is the generic structure for the definition of rulesets.
- Provides connection tracking as well as NAT/IP Masquerading

What can be done with it?

- Stateful(ipV4 only)/Stateless packet filtering
- Connection Tracking
- NAT and NAPT
- QoS routing (not handled directly w/iptables, but using packet marking through netfilter)
- Packet Mangling (alteration of headers, etc)

In other words ...

- Iptables is a highly configurable:
  - Firewall
  - and a NAT box

- Small routers at home often contain linux and iptables
- Firewall systems are often built as linux/iptables system
- There are many GUI-s for managing iptables rules, i.e. easier configuration: these GUI-s are not firewalls, just user-friendly interfaces for setting firewall rules
- There is a basic rule-setting command in linux: **iptables**
- User-friendly GUI-s can be avoided and **iptables** command used instead

How to use IPTABLES?

- First, necessary options must be compiled into the linux kernel
- Second, the iptables package must be installed
- Third, start adding rules (as a superuser), giving command line commands like:

  iptables -I INPUT -p tcp --dport 22 -j ACCEPT

- Observe: **iptables** is a name of the program adding the rule to the iptables rulebase
- Linux kernel then uses the rulebase when processing IP packets

What happens inside?

- When a packet first enters the firewall, it hits the hardware and then gets passed on to the proper device driver in the kernel.
- Then the packet starts to go through a series of steps in the kernel, before it is either sent to the correct application (locally), or forwarded to another host - or whatever happens to it.

Chains and policies

- There are three main “chains”:
  - input
  - forward
  - output
- For each chain you must first give a general policy:
  - accept or
  - drop
- Then you start adding exceptions to policies

Enabling Options in the Kernel

Networking->

Networking Options->

Network packet filtering (replaces ipchains) --->

IP: Netfilter Configuration --->

It is safe to enable all modules, I recommend compiling the ftp and other connection tracking modules as modules and not into the kernel so that you can verify they are loaded and functioning.

Basic iptables syntax

- iptables --flush
- iptables --policy INPUT DROP
- iptables --policy OUTPUT DROP
- iptables --policy FORWARD DROP
- iptables -A INPUT -i lo -j ACCEPT
- iptables -A OUTPUT -o lo -j ACCEPT

First example script

```
#!/bin/bash
set -x
# Load needed kernel modules
modprobe ip_conntrack
modprobe ip_conntrack_ftp
# Clear any existing firewall stuff before we start
iptables --flush
iptables -t nat --flush
iptables -t mangle --flush
# As the default policies, drop all incoming traffic but allow all
# outgoing traffic.  This will allow us to make outgoing connections
# from any port, but will only allow incoming connections on the ports
# specified below.
iptables --policy INPUT DROP
iptables --policy OUTPUT ACCEPT
```

Second example script

```
#!/bin/bash
# flush all chains
iptables -F
# set the default policy for each of the pre-defined chains
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
# allow establishment of connections initialised by my outgoing packets
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
# drop everything else
iptables -A INPUT -i eth+ -p udp -j DROP
iptables -A INPUT -i eth+ -p tcp -m tcp --syn -j DROP
# accept anything on localhost
iptables -A INPUT -i lo -j ACCEPT
```

.... continue

Allow all incoming traffic if it is coming from the local loopback device

iptables -A INPUT -i lo -j ACCEPT

Related and established connections: see
http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html
Accept all incoming traffic associated with an established
connection, or a "related" connection
This will automatically handle incoming UDP traffic associated with
DNS queries, as well as PASSIVE mode FTP (provided the
ip_conntrack_ftp module is loaded)

iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT

... continue

Allow connections on selected ports to the firewalled computer:

22 ssh, 80 web, 25 smtp (mail)

iptables -A INPUT -p tcp -i eth0 --dport 22 -m state --state NEW -j ACCEPT

iptables -A INPUT -p tcp -i eth0 --dport 80 -m state --state NEW -j ACCEPT

iptables -A INPUT -p tcp -i eth0 --dport 25 -m state --state NEW -j ACCEPT

Allow icmp input so that people can ping us

iptables -A INPUT -p icmp -j ACCEPT

... continue

Logging: first, eliminate any packets that are going to broadcast
addresses, since they will overwhelm the log files if there are any
windows computers on our network. Also, don't log pesky multicast
packets that we block.

iptables -A INPUT -d 255.255.255.255/0.0.0.255 -j DROP
iptables -A INPUT -d 224.0.0.1 -j DROP

Log all other blocked packets, and change DROP to REJECT to be
polite and allow people connecting to a blocked port to receive a
"connection refused" message instead of timing out after 30 seconds.

iptables -A INPUT -j LOG
iptables -A INPUT -j REJECT

Third example script (I use)

```
#!/bin/sh
# firewall.sh - Configurable per-host firewall for workstations and
# servers.(c) 2003 Tero Karvinen - tero karvinen at iki fi - GPL
# Cleanup old rules
 # All the time firewall is in a secure, closed state

iptables -P INPUT DROP
iptables -P FORWARD DROP

iptables --flush       # Flush all rules, but keep policies
iptables --delete-chain
```

... continued

Workstation Minimal firewall

iptables -P FORWARD DROP

iptables -P INPUT DROP

iptables -A INPUT -i lo --source 127.0.0.1 --destination 127.0.0.1 -j ACCEPT

iptables -A INPUT -m state --state "ESTABLISHED,RELATED" -j ACCEPT

iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT

iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT

iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT

... finishing up

####### HOLES ####### Edit holes below, then run this script again

#iptables -A INPUT -p tcp --dport ssh -j ACCEPT

#iptables -A INPUT -p tcp --dport http -j ACCEPT

#iptables -A INPUT -p tcp --dport https -j ACCEPT

####################### Edit above

iptables -A INPUT -j LOG -m limit --limit 40/minute

iptables -A INPUT -j DROP

Save

iptables-save > /etc/sysconfig/iptables

echo ": Done."

Another simple example

Scenario: Personal Firewall that should allow all outbound connections and restrict all inbound traffic to just an SSH server running on port 22.

Simple Example

Initial Rules:

iptables -L

Chain INPUT (policy ACCEPT)

target prot opt source destination

Chain FORWARD (policy ACCEPT)

target prot opt source destination

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

Simple Example

Lets add the rule to allow incoming tcp connections on 22:

iptables -I INPUT -p tcp --dport 22 -j ACCEPT
iptables –L

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Simple Example

Lets add a rule to allow incoming connections from ourself (on the loopback interface only – outside interface could be spoofed).

iptables -I INPUT -i lo -j ACCEPT

iptables –A INPUT –i eth0 –s localhost –j DROP

iptables –L

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
DROP       all  --  localhost            anywhere
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Simple Example

Lets add a rule to allow incoming connections on any port if it is already established or related to an established connection.

iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables –L

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Simple Example

Now lets add a rule to log all packets before we drop them:

iptables -A INPUT -j LOG --log-level debug --log-prefix "IPTABLES: DROPPED"
iptables –L

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
LOG        all  --  anywhere             anywhere            LOG level debug prefix `IPTABLES: DROPPED'

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Simple Example

Now our drop rule, we have a choice here we can either drop the packet (black holes it) or we can reject the packet, which sends a response to the sender.

My preference is to drop the packet, as it can signify that there is no host attached to the address, but it can interfere with programs that rely on a reject.

Simple Example

iptables -A INPUT -j DROP

iptables –L

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
LOG        all  --  anywhere             anywhere            LOG level debug prefix `IPTABLES: DROPPED'
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Something More Complex

When creating a more involved firewall ruleset, it is a good idea to write down all the functionality that you require before starting. Make sure that you identify all services, their ports, and the address that need access.

Something More Complex

Scenario: Linux machine will function as router, 2 separate subnets. Eth0 is the WAN interface, Eth1 connects to subnet 10.1.1.0/24, Eth2 connects to subnet 192.168.2.0/24.

We wish to provide NAT to the two subnets, both subnets should be able to freely communicate with each other and the server.

The server will be running an http server on ports 80 and 443, an ssh server on port 22. We will be running an ftp server on the router, port 21, that will require passive ftp access (no SSL) and active ftp access. Port 8080 needs to be forwarded to machine 10.1.1.2 for a web program.

Something More Complex

```
iptables -I INPUT -s localhost -i eth0 -j DROP
iptables -A INPUT -m tcp -m match multiport --dport 22,21,80,443 -j ACCEPT
iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i eth0 -p udp -m udp multiports --dports 57,63 -j REJECT --reject-with icmp-port-unreachable
iptables -A INPUT -i eth0 -p tcp -s 0/0 --sport 1024:65535 --dport 1024:65535 -m state --state
    ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -p icmp -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -j LOG --log-prefix "[INPUT DROPPED]:" --log-level debug
iptables -I FORWARD -d 10.1.1.0/24 -i eth2 -j ACCEPT
iptables -A FORWARD -d 192.168.2.0/24 -i eth1 -j ACCEPT
iptables -A FORWARD -s 10.1.1.0/24 -i eth1 -j ACCEPT
iptables -A FORWARD -s 192.168.2.0/24 -i eth2 -j ACCEPT
iptables -A FORWARD -j LOG --log-prefix "[FORWARD DROPPED]:" --log-level debug
iptables -P FORWARD DROP
iptables -t nat -I POSTROUTING -o eth2 -j MASQUERADE
iptables -t nat -I POSTROUTING -o eth1 -j MASQUERADE
iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE
iptables -t nat -I PREROUTING -i eth0 -p tcp -m tcp --dport 8080 -j DNAT --to-destination 10.1.1.2
```